Lambek theorem proving and feature unification

van der Linden, H.J.B.M.

Publication date:
1989

Document Version
Publisher's PDF, also known as Version of record

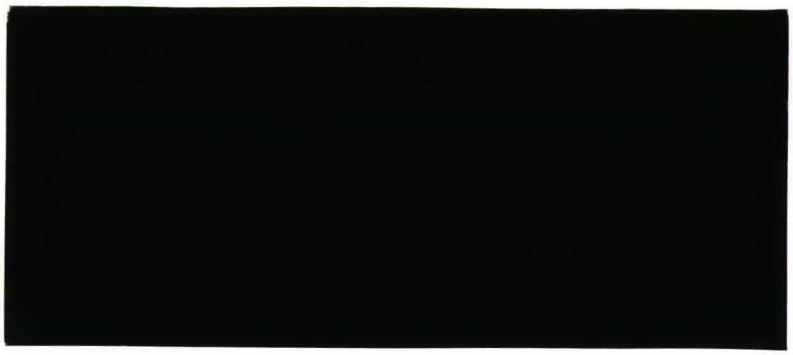
[redacted]

ITK

RESEARCH REPORT

INSTITUTE FOR LANGUAGE TECHNOLOGY AND ARTIFICIAL INTELLIGENCE

INSTITUUT VOOR TAAL- EN KENNISTECHNOLOGIE



ITK Research Report No. 6

April 1989

Lambek Theorem Proving and Feature Unification

Erik-Jan van der Linden

To appear in: Proceedings of the European Chapter of the ACL. Manchester, UK, 10-12 April 1989.

Institute for Language Technology and Artificial Intelligence ITK

Tilburg University, Tilburg, the Netherlands

LAMBEK THEOREM PROVING AND FEATURE UNIFICATION

Erik-Jan van der Linden*
Institute for Language Technology and Artificial Intelligence

Tilburg University
PO Box 90153, 5000 LE Tilburg, The Netherlands

1 ABSTRACT

Feature Unification can be integrated with Lambek Theorem Proving in a simple and straightforward way. Two principles determine all distribution of features in LTP. It is not necessary to stipulate other principles or include category-valued features where other theories do. The structure of categories is discussed with respect to the notion of category structure of Gazdar et al. (1988).

2 INTRODUCTION

A tendency in current linguistic theory is to shift the 'explanatory burden' from the syntactic component to the lexicon. Within Categorial Grammar (CG), this so-called lexicalist principle is implemented in a radical fashion: syntactic information is projected entirely from category structure assigned to lexical items (Moortgat, 1988). A small set of rules like (1) constitutes the grammar. The rules reduce sequences of categories to one category.

(1) X:a X\Y:b => Y:b(a)

CG implements the Compositionality Principle by stipulating a correspondence between syntactic operations and semantic operations (Van Benthem 1986).
An approach to the analysis of natural language in CG is to view the categorial reduction system, the set of reduction rules, as a calculus, where parsing of a syntagm is an attempt to prove that it follows as a theorem from a set of axioms and inference rules. Especially by the work of Van Benthem (1986) and Moortgat (1988) this view, which we will name with Moortgat (1987a) Lambek Theorem Proving (LTP; Lambek, 1958), has become popular among a number of linguists.
The descriptive power of LTP can be extended if unification (Shieber, 1986) is added. Several theories have been developed that combine categorial formalisms and unification based formalisms. Within Unification Categorial Grammar (UCG, Calder et al., 1988, Zeevat et al., 1986) unification "is the only operation over grammatical objects" (Calder et al. 1988, p. 83), and this includes syntactic and semantic operations. Within Categorial Unification Grammar (Uszkoreit, 1986; Bouma, 1988a), reduction rules are the main operation over grammatical objects, but semantic operations are reformulated within the unification formalism, as properties of lexemes (Bouma et al., 1988). These formalisms thus lexicalize semantic operations.
The addition of unification to the LTP formalism described in this paper maintains the rules of the syntactic *and* semantic calculus as primary operations, and *adds* unification to deal with syntactic features only. We will refer to this addition as Feature Unification (FU), and we will call the resulting theory LTP-FU.
In this paper firstly the building blocks of the theory, categories and inference rules, will be described. Then two principles will be introduced that determine the distribution of features, not only for the rules of the calculus, but also for reduction rules that can be derived within the calculus. From the discussion of an example it is concluded that it is not necessary to stipulate other principles or include category-valued features where other theories do.

*Part of the research described in this paper was carried out within the 'Categorial Parser Project' at ITI-TNO. I wish to thank the people whom I had the pleasure to cooperate with within this project: Brigit van Berkel, Michael Moortgat and Adriaan van Paassen. Gosse Bouma, Harry Bunt, Bart Geurts, Elias Thijsse, Ton van der Wouden, and three anonymous ACL reviewers made stimulating comments on earlier versions of this paper. Michael Moortgat generously supplied a copy of the interpreter described in his 1988 dissertation

3 CATEGORIES

In LTP categories and a set of inference rules constitute the calculus. The addition of FU necessitates the extension of these with respect to LTP without FU. Categories are for a start defined in the framework introduced by Gazdar et al. (1988). Gazdar et al. define category structure on a metatheoretical level as a pair $< \Sigma, C >$. Σ is a quadruple$<F, A, \tau, \rho>$ where F is a finite set of features; A is a set of atoms; τ is a function that divides the set of features into two sets, those that take atomic values (Type 0 features), and those that take categories as values (Type 1). ρ is a function that assigns a range of atomic values to each Type 0 feature. C is a set of constraints expressed in a language L_c. The reader is referred to Gazdar et al. (1988) for a precise definition of this language: we will merely use it here. For LTP-FU, the category structure in (2) and the constraints in (3) apply.

(2)

F = { DOMAIN, RANGE, FIRST, LAST, CONNECTIVE, LABEL} ∪ FEAT_NAMES
FEAT_NAMES = {PERSON,...., TENSE}
A = BASCAT ∪ CONNECTIVES ∪ FEAT_VALUES
BASCAT = { N, V,...}
CONNECTIVES = {/,\,*}
FEAT_VALUES = {1,2,3,.....}
τ = { <DOMAIN, 1>, <RANGE, 1>, <FIRST, 1>, <LAST, 1>, <CONNECTIVE,0>,...}
ρ = { <CONNECTIVE, CONNECTIVES>, <LABEL, BASCAT>, <PERSON, {1,2,3,}>,...}

(3)

(a) □(CONNECTIVE ↔ ¬ LABEL)
(b) □(DOMAIN ↔ RANGE)
(c) □(DOMAIN ↔ CONNECTIVE:(/ ∨ \))
(d) □(FIRST ↔ CONNECTIVE:*)
(e) □(FIRST ↔ LAST)
(f) □(RANGE:f → f ∉ FEAT_NAMES)

The fact that 'category' is a central notion in CG justifies the division between features that express syntactic combinatorial possibilities ({DOMAIN,..., LABEL}) and other features (FEAT_NAMES) in (2) [1].
In what follows we will use 'feature structure' to denote a set of feature-value combinations with features from FEAT_NAMES. We will use 'category' in the sense common in categorial linguistics. For a category with feature structure, we will use the term 'category specification'.
Constraint (3)(a) ensures that a category is either complex or basic. Functor categories, those with the connective \ or / are specified by (3)(b), (3)(c); other complex categories are specified by (3)(d) and (e); (3)(f) describes the distribution of features from FEAT_NAMES. Here we follow Bouma (1988a) in the addition of features to complex categories. Firstly features are added to the argument (DOMAIN) in a complex category. This is "to express all kinds of subcategorization properties which an argument has to meet as it functions as the complement of the functor" (Bouma, 1988a, p. 27). Secondly, the category as a whole, rather than the RANGE carries features. "This has the advantage that complex categories can be directly characterized as finite, verbal etc." (Bouma, 1988a, p. 27; cf. Bach, 1983).

[1] This view can for instance be found in the following citation from Calder et al. (1986): "(..) these [categories] can carry *additional* feature specifications" (Calder et al., 1986, p. 7; my emphasis).

4 INFERENCE-RULES

A sequent in the calculus is denoted with P => T, where P, called the antecedent, and T, the succedent, are finite sequences of category specifications: P = K_1 ... K_m and T = L. In LTP P and T are required to be non-empty; notice that the succedent contains one and only one category specification. The axioms and inference rules of the calculus define the theorems of the categorial calculus. Recursive application of the inference rules on a sequent may result in the derivation of a sequent as a theorem of the calculus.
In what follows, X, Y and Z are categories; A,B,C,D and E are feature structures; K,L,M,N are category specifications; P, T, Q, U, V are sequences of category specifications, where P, T and Q are non-empty. We use the notation *category;feature structure:semantics.*

Axioms are sequents of the form X;A:a => X;A:a. Note that identical letters for categories and semantic formulas denote identical categories and identical semantic formulas; identical letters for feature structures mean unified feature structures; and identical letters for category specifications mean category specifications with identical categories and unified features structures. From the form of the axiom it may follow that feature structures in antecedent and succedent should unify. This principle is the Axiom Feature Convention (AFC).
In (4) the inference rules of LTP-FU are pre-

sented [2]. $[\backslash - e]$ denotes a rule that *e*liminates a \-connective. *i* denotes *i*ntroduction. The 'active type' in a sequent is the category from which the connective is removed.

(4)

```
[/-e] U,(X/Y;A);B:b,T,V => Z
      ----------------------
      if  T => Y;A:a
      and U,X;B:b(a),V => Z

[\-e] U,T,(Y;A\X);B:b,V => Z
      ----------------------
      if  T => Y;A:a
      and U,X;B:b(a),V => Z

[*-e] U, K:a*L:b,V => M
      -----------------
      if  U,K:a,L:b,V => M

[/-i] T => (X/Y;A);B:^v.b
      -------------------
      if  T,Y;A:v => X;B:b

[\-i] T => (Y;A\X);B;^v.b
      -------------------
      if  Y;A:v, T => X;B:b

[*-i] P:a,Q:b => K*L:c*d
      ------------------
      if  P:a => K:c
      and Q:b => L:d
```

Certain feature structures are required to unify in inference rules. We formulate the so-called Active Functor Feature Convention (AFFC) to control the distribution of features. This convention is comparable to Head Feature Convention (Gazdar et al., 1985) and Functor Feature Convention (Bouma, 1988a). The AFFC states that the feature structure of an active functor type must be unified with the feature structure on the RANGE of the functor in the subsequent.

5 AN EXAMPLE

This paragraph limits itself to some observations concerning reflexives because this sheds light on a remaining question: are there principles other than AFFC and AFC necessary to account for 'FOOT' phenomena?

There are two properties of reflexive pronouns that have to be accounted for in the theory. Firstly, the reflexive pronoun has to agree in number, person, and gender with some antecedent in the sentence (Chierchia, 1988), mostly the subject. Secondly, the reflexive pronoun is not necessarily the head of a constituent (Gazdar et al., 1985).

The HFC in GPSG (Gazdar et al., 1985) cannot instantiate the antecedent information of a reflexive pronoun on a mothernode in cases where the reflexive is not the head of a constituent. Therefore in GPSG the so-called FOOT Feature Principle (FFP) is formulated. Together with the Control Agreement Principle (CAP) and the HFC, the FFP ensures that agreement between the demanded antecedent and the reflexive pronoun is obtained. Inclusion of a principle similar to FFP, and the use of category-valued features could be a solution for CUG. However, a solution that makes use of means supplied by categorial theory would keep us from 'stipulating axioms and principled', and as we will see, has as a consequence that we can avoid the use of category-valued features.

For an account of reflexives in LTP-FU we will make use of reduction laws, other than the inference rules in (4). These reduction laws (like 1) normally have to be stipulated within categorial theory, but in LTP they can be derived as theorems within the calculus presented in (4) (Moortgat, 1987b). Feature distribution for these laws in LTP-FU can also be derived within the calculus with the application of AFFC and AFC and thus feature unification within these reduction laws also falls out as 'theorem' of the calculus: it is not necessary to include other principles than AFFC and AFC. In (5) a derivation for the reduction law *composition* is given (cf. Moortgat, 1987, p. 6).

(5)

```
[COMP]
(X/Y;A);D (Y/Z;B);A  => (X/Z;B);D
_________________________________[/-i]
if (X/Y;A);D (Y/Z;B);A Z;B => X;D
   ______________________________[/-e]
   if  Z;B            => Z;B
   and (X/Y;A);D Y;A => X;D
        ___________________[/-e]
        if  Y;A     =>Y;A
        and X;D     =>X;D
```

(6)

```
[CUT]
U    T   V   =>  L
------------------
if   T   =>  K:a
and  U   K:a V=> L
```

[2] To envisage the rules without FU, just leave out all feature structures

(7)

(a)

Jan houdt van zichzelf.
John loves of himself.

(b)

zichzelf: (((np;3S\s)/np;C);A \ (np;3S\s));A

(c)

```
houdt                          van
((np;3S\s)/pp;A);B  (pp/np;C);D
-------------------------------------------[COMP]
((np;3S\s)/np;C);B
```

(d)

```
Jan   houdt                          van          zichzelf
np;3S ((np;3S\s)/pp;A);B (pp/np;C);D (((np;3S\s)/np;C);A\(np;3S\s));A => s;E
-------------------------------------------------------------------------[CUT]
np;3S ((np;3S\s)/np;C);B (((np;3S\s)/np;C);A\(np;3S\s));A => s;E
-----------------------------------------------------------[\-e]
if ((np;3S\s)/np;C);B => ((np;3S\s)/np;C);A
and np (np;3S\s);A => s;E
    ---------------------------------[\-e]
    if np;3S => np;3S
    and s => s;E
```

(e)

```
^x^yHOUDT(x)(y) ^z.VAN(z)
-------------------------[COMP]
^z^yHOUDT(VAN(z))(y)
```

(f)

```
Jan  houdt            van       zichzelf
JAN  ^x^yHOUDT(x)(y) ^z.VAN(z) ^h^fh(f)(f)
-----------------------------------------[CUT]
JAN  ^z^yHOUDT(VAN(z))(y) ^h^fh(f)(f)
         ---------------------------[\-e]
         ^f.HOUDT(VAN(f))(f)
-----------------------------[\-e]
HOUDT(VAN(JAN))(JAN)
```

The cut rule (6) is not an inference rule, but a structural rule that is used to include proofs from a 'data base' into other proofs, for instance to include the results of the application of composition to part of a sequent. The cut rule is added to the inference rules of the calculus [3]. In (7(d)) the cut rule is used once to include a partial proof derived with the composition rule. The lexical category we assume the reflexive to have (see 7(b)) takes a verb with two arguments as its argument, and results in a verb with one argument. The verb requires, in the example, its subject to carry two feature-value pairs: [num#sing,pers#3]. (In (7(d)), all feature structures containing these features are abbreviated with the notation 3S.) These features are instantiated for the subject of the resulting one-argument verb. (7) gives a derivation where the reflexive is embedded in a prepositional phrase. In the example only relevant feature structures have been given actual feature-value pairs. (7(b)) presents the category of the reflexive. (c) presents one reduction using the composition rule and (d) presents the reduction of the whole sequent. The derivation of the semantic structure is presented seperately (e-f) from the syntactic derivation to improve readability.
The reflexive's semantics imposes equality upon the arguments of the verb (Szabolcsi, 1987; but see also Chierchia (1988) and Popowich (1987) for other proposals). Note that in all cases, the reflexive should combine with the verb before the subject comes into play: the reflexive's semantics can only deal with λ-bound variables as arguments.

6 IMPLEMENTATION

In this section a Prolog implementation of LTP-FU is described. The implementation makes use of the interpreter described in Moortgat (1988). Categorial calculi, described in the proper format, can be offered to this interpreter. The interpreter then uses the axioms, inference rules and reduction rules as data and applies them to an input sequent recursively, in order to see whether the input sequent is a theorem in the calculus. In order to 'implement' a calculus, firstly it has to be described in a proper format. => and ← are defined as Prolog operators and denote respectively *derivability* in the calculus and *inference* during theorem proving. So, for instance with respect to the axiom, we may say that we have shown that X;A reduces to X;B if feat_des_unify between A and B holds and true holds. The list notation is equal to the usual Prolog list notation, and is used to find the proper number of arguments while unifying an actual sequent with a rule. For instance $[T|R]$ cannot be instantiated as an empty list, whereas U can be instantiated as one. The LTP-FU calculus is presented in (8) (semantics is left out for readability).

(8)

```
[axiom] [X;A] => [X;B] <-
        (feat_des_unify(A,B)) &
        true.

[/-e]   (U,[(X/Y;A);B],[T|R],V) => [Z]<-
           [T|R] => [Y;A] &
           (U,[X;B],V) => [Z].

[\-e]   (U,[T|R],[(Y;A\X);B],V) => [Z]<-
           [T|R] => [Y;A] &
           (U,[X;B],V) => [Z].

[*-e]   (U, [K*L],V) => [M]          <-
           (U,[K,L],V) => [M].

[/-i]   [T|R] => [(X/Y;A);B]         <-
           [T|R],[Y;A] => [X;B].

[\-i]   [T|R] => [(Y;A\X);B]         <-
           Y;A, [T|R] => [X;B].

[*-i]   ([P|R],[Q|R1) => [K*L]       <-
           [P|R]  => [K] &
           [Q|R1] => [L].
```

Note that feature unification is added explicitely: identity statements are interpreted "as instructions to replace the substructures with their unifications" (Shieber, 1986, p. 23). Prolog, however, does not allow this so-called destructive unification and therefore unification is reformulated. The necessity for destructive unification becomes clear from (9), where it is necessary to let features percolate to the "mother node" of a constituent. Note that in (9) reentrance for the modifier *het* and the specifier *kleine* is necessary (cf. Bouma, 1988a) to let the feature-value pair sex#fem percolate to the np. Reentrance is denoted with a number followed by a hook. It is represented *within* lexical items; it is therefore not necessary to stipulate principles to account for percolation through reentrance.

[3] For consequences of the addition of this rule, see Moortgat (1988)

(9)

het	kleine	meisje
the	little	girl

(np/n;1>C);1>D (n/n;2>A);2>B n;[sex#fem]

Within the ITI-TNO parser project (see footnote on first page), an attempt is made to develop a parser based on the mechanisms described here, using standard software development methods and techniques. During the so-called information analysis and the design stage (Van Berkel et al., 1988), several prototypes of a Lambek Theorem Prover have been developed (Van Paassen, 1988). Implementation in C is currently undertaken, including semantic representation. Addition of Feature unification to this parser is scheduled for 1989. Lexical software for this purpose (in C) is available (Van der Linden, 1988b).

7 CONCLUDING REMARKS

Feature unification can be added to LTP in a simple and straightforward way. Because reduction laws that fall out (including feature unification) as theorems in LTP-FU can account for FOOT phenomena, it is not necessary to 'stipulate' category-valued FOOT features and mechanisms to account for their percolation. Not only reflexives, but also unbounded dependencies can be described without the use of category-valued features. Bouma (1987) shows that the addition of Type 0 features GAP with BASCAT as its value and ISL with {+,-} as its value are the features used in an account of unbounded dependencies [4].

LTP-FU can do without category-valued features in FEAT_NAMES, and this obviously reduces complexity of the unification process. We can add to this that it is possible to develop efficient algorithms and computerprograms for LTP (Moortgat, 1987a; Van der Wouden and Heylen, 1988; Van Paassen, 1988; Bouma, 1989). Therefore LTP-FU is attractive for computational linguistics.

A problem remains with respect to the semantics of reflexives we assume here. A reflexive as *zichzelf* in (7) can only take a verb as an argument, and not for instance a combination of a subject and a verb (S/NP): the reflexive only operates on a functor with two different λ-bound arguments. This implies that it is hard for this kind of category to participate in a Left-to-Right analysis (Ades and Steedman, 1982). A solution could be to describe reflexives syntactically as functors of type (X/NP)\X, that impose reentrance (and not equality) upon the NP argument and some other NP. This implies however that we should not only construct a semantic representation, but also a representation of the syntactic derivation, in order to be able to refer to NP's that have already served as arguments to some functor. Future research will be carried out with respect to this *constructive* categorial grammar.

A final remark concerns the notion of category structure taken from Gazdar et al. (1988) and applied here. For an account of modifiers and specifiers, it is necessary to include reentrant features. Therefore the definition of category structure in LTP-FU, but also that in CUG and UCG where reentrance is used as well, necessitates extended versions of the notion Gazdar et al. supply.

[4] Van der Linden (1988a) discusses S-V agreement.

8 LITERATURE

Ades, A.; and Steedman, M. 1982 On the order of words. *Linguistics and Philosophy*, 4, pp. 517-558.

Bach, E. 1983 On the relationship between word-grammar and phrase-grammar. *Natural Language and Linguistic Theory* 1, 65-89.

van Benthem, J. 1986 Categorial Grammar. Chapter 7 in Van Benthem, J., Essays in Logical Semantics. Reidel, Dordrecht.

van Berkel, B.; van der Linden, H.; and van Paassen, A. 1988 Parser Project, analysis and design. Internal report 88 ITI B 24, ITI-TNO, Delft (Dutch).

Bouma, G. 1987 A unification-based analysis of unbounded dependencies in categorial grammar. In: Groenendijk et al. 1987. pp. 1-19.

Bouma, G. 1988a Modifiers and specifiers in categorial unification grammar. *Linguistics* 26, 21-46.

Bouma, G. 1989 Efficient processing of flexible categorial grammar. This volume.

Bouma, G.; König, E.; Uszkoreit, H. 1988 A flexible graph-unification formalism and its application to natural-language processing. *IBM Journal of Research and Development*, 32, pp 170-184.

Calder, J.; Klein, E.; and Zeevat, J. 1988 Unification categorial grammar: a consise, extendable grammar for natural language processing. In *Proceedings of COLING '88*, Budapest.

Chierchia, G. 1988. Aspects of a categorial theory of binding. In Oehrle et al. 1988. pp. 125-151.

Gazdar, G.; Klein, E.; Pullum, G.; and Sag, I. 1985 *Generalized Phrase Structure Grammar.*

Basil Blackwell, Oxford.
Gazdar, G.; Pullum, G.; Carpenter, R.; Klein, E.; Hukari, T.; and Levine, D. 1988 Category Structure. *Computational Linguistics* 14, 1-19.
Groenendijk, J.; Stokhof, M.; and Veltman, F., Eds. 1987 *Proceedings of the sixth Amsterdam Colloquium.* April 13-16 1987. University of Amsterdam: ITLI.
Lambek, J. 1958 The mathematics of sentence structure. *Am. Math. Monthly* 65, 154-169.
Klein, E.; and Van Benthem, J., Eds. 1988. *Categories, Polymorphism and Unification.* Edinburgh.
van der Linden, H. 1988a GUACAMOLE, Grammatical Unification-based Analysis in a CAtegorial paradigm with MOrphological and LExical support. Internal report 88 ITI B 37, ITI-TNO, Delft (Dutch).
van der Linden, H. 1988b User-documentation for SIMPLEX. Internal report 88 ITI B 34, ITI-TNO, Delft (Dutch).
Moortgat, M. 1987a Lambek Theorem Proving. In Klein; and van Benthem 1988, pp. 169-200.
Moortgat, M. 1987b Generalized Categorial Grammar. To appear in Droste, F., Ed., *Mainstreams in Linguistics.* Benjamins, Amsterdam.
Moortgat, M. 1988 *Categorial Investigations. Logical and linguistic aspects of the Lambek calculus.* Dissertation, University of Amsterdam.
Oehrle, R.; Bach, E.; and Wheeler, D. Eds., 1981 *Categorial grammar and natural language structure.* Reidel, Dordrecht.
Van Paassen, A. 1988 Reduction of the searchspace in Lambek Theorem Proving. Internal report 88 ITI B 23, ITI-TNO, Delft (Dutch).
Popowich, F. 1988, A Unification-Based Framework for Anaphora in Klein and van Benthem 1988. pp. 277-305.
Shieber, S. 1986 *An introduction to Unification-Based Approaches to Grammar.* University of Chicago Press, Chicago.
Szabolcsi, A. 1987 Bound variables in syntax (are there any?). In Groenendijk et al. 1987, pp. 331-351.
Uszkoreit, H. 1986 Categorial Unification Grammars. In *Proceedings of COLING 1986*, Bonn.
van der Wouden, T.; and Heylen, D. 1988 Massive Disambiguation of large text corpora with flexible categorial grammar. In *Proceedings of COLING 1988*, Budapest.
Zeevat, H.; Klein, E.; and Calder, J. 1986 Unification Categorial Grammar. Paper, University of Edinburgh.

ITK: P.O. BOX 90153 5000 LE TILBURG THE NETHERLANDS